VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vytvoření Add-inu pro Visual Studio 2010

Creating Add-in for Visual Studio 2010

2013 Jiří Hanzelka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Hanzelka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vytvoření Add-inu pro Visual Studio 2010
Creating Add-in for Visual Studio 2010

Zásady pro vypracování:

Paralelní aplikace napsané v jazyce C#, lze obtížně ladit ve vývojovém prostředí Visual Studio 2010, protože toto prostředí standardně nepodporuje vývoj paralelních aplikací v tomto jazyce. Cíle této práce jsou následující:

1. Seznamte se s možnostmi práce s procesy ve Visual Studiu 2010.
2. Seznamte se s principy tvorby Add-inu ve Visual Studiu 2010.
3. Proveďte analýzu a návrh Add-inu pro automatické spuštění ladění paralelní aplikace napsané v C#.
4. Implementujte daný Add-in.

Seznam doporučené odborné literatury:

Keyvan Nayyeri, Professional Visual Studio Extensibility. Wrox (March 10, 2008). ISBN-13 978-0470230848.
Lars Powers, Microsoft Visual Studio 2008 Unleashed. Sams; 1 edition (June 9, 2008). ISBN-13 978-0672329722.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lukáš Vojáček**

Datum zadání: 18.11.2011
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 .....

Rád bych na tomto místě poděkoval panu Ing. Lukáši Vojáčkovi, který mi s touto prací pomáhal.

Abstrakt

Ladění paralelních aplikací napsaných v jazyce C# je ve vývojovém prostředí Visual Studio 2010 složité. Toto vývojové prostředí je ale možné rozšiřovat pomocí takzvaných Add-inů. V této práci se budu zabývat tvorbou Add-inu, který umožní lépe ladit paralelní aplikace. Tento Add-in bude vyvíjen pro Visual Studio 2010 a 2012.

Klíčová slova: Add-in, Visual Studio, C#, HPC

Abstract

Debugging parallel applications written in C# language is in the Visual Studio 2010 development environment complicated. But this development environment can be extended by so called Add-ins. In this thesis, I will deal with creation of Add-in that will allow better debug parallel applications. This Add-in will be developed for Visual Studio 2010 and 2012.

Keywords: Add-in, Visual Studio, C#, HPC

Seznam použitých zkratek a symbolů

HPC	–	High Performance Computing
XML	–	Extensible Markup Language
DLL	–	Dynamic Link Library
ID	–	Identifier
GUID	–	Globally Unique Identifier
SDK	–	Software Development Kit
GUI	–	Graphical User Interface
MEF	–	Managed Extensibility Framework
IDE	–	Integrated Development Environment
VSIX	–	Visual Studio Extension
MSI	–	Microsoft Installer
VSI	–	Visual Studio Content Installer
GAC	–	Global Assembly Cache

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Visual Studio je vývojové prostředí od společnosti Microsoft. Toto prostředí slouží k vývoji mnoha druhů aplikací, at'už to jsou konzolové, webové, mobilní a jiné aplikace. Při tvorbě tohoto prostředí se bralo ohled na to, aby jej bylo možné jednoduše rozšiřovat. Bylo navrženo několik způsobů, kterými je možné toto prostředí rozšířit. Každý způsob má určité výhody. Třeba makra slouží především k automatizaci určitých činností ke zrychlení práce s Visual Studiem. Nebo balíčky(VSPackages) umožňují přidávat podporu jiných programovacích jazyků do tohoto vývojového prostředí. Tyto balíčky jsou ale složitější k vytvoření, protože více zasahují do Visual Studia a mají tak více možností. Jednou možností jak toto prostředí rozšířit, je pomocí takzvaných add-inů. Add-iny jsou vhodné k tvorbě různých nástrojů, které umožní přidávat nové funkce.

V první části této bakalářské práce se zmíním o možnostech rozšíření Visual Studia a kde tato rozšíření najít. Především se ale zabývám addi-ny. Jak tyto add-iny vytvářet, jestli se liší jejich ladění od jiných programů. Jak je spravovat z Visual Studia. Dále se zmíním o možnostech nasazení těchto add-inů a jejich publikací.

Druhá část této práce je o tvorbě specifického add-inu, který zjednoduší ladění paralelních aplikací napsaných v jazyce C#. Zmíním se zde o balíku nástrojů HPC Pack 2008 SDK, který umožňuje vytvářet paralelní aplikace díky simulaci HPC clusteru na jednom PC. Dále popíšu základní kroky, které add-in musí zvládat s výpisy kódu, jak se toho dá dosáhnout, problémy těchto kroků a možnosti jejich řešení. Potom zde bude samotný návrh tohoto add-inu a jeho GUI. Na konci této části vytvořím pro daný add-in instalátor pro jednoduché nasazení a publikaci.

2 Visual Studio

Visual Studio je integrované vývojové prostředí IDE od firmy Microsoft, které obsahuje nástroje k vývoji, návrhu a ladění aplikací a služeb. Umožňuje vývoj aplikací na mobilní zařízení s Windows Phone, PC, web, ale i Cloud s platformou Windows Azure.

Microsoft vydává několik edicí Visual Studia, základní jsou Professional, Premium a Ultimate. Potom jsou ještě bezplatné verze Express, které neobsahují pokročilejší funkce, ale pro studentské účely nebo vývoji jednoduchých aplikací dostačují.

V průběhu psaní bakalářské práce vyšla i verze Visual Studio 2012, která umožňuje vývoj Modern UI (dříve Metro) aplikací pro Windows 8. Vzhled této verze se snaží přiblížit vzhledu Windows 8, ve kterém již není použito Windows Aero, tedy průhledné okna . Barvy tohoto prostředí proto používají především různé odstíny šedé barvy.

2.1 Visual Studio 2010

Ceny těchto edic se pohybují přibližně od 10 do 300 tisíc korun za standardní verzi. Koupit se dají různé verze těchto edic, které mohou být výhodnější pro velké nebo malé firmy, kvůli ceně jsou především vhodné multilicence.

Professional

Tato edice je základní mezi placenými edicemi. Umožňuje psát aplikace ve všech jazycích patřících do .NET. V této verzi je samozřejmě možné ladit a využívat testování kódu. Tato edice je tvořena především pro samostatné vývojáře.

Premium

Je to další edice v pořadí. Obsahuje vše co edice Professional a navíc přidává pokročilé nástroje pro vývoj, jako jsou automatizované testy, analýza kódu. Umožňuje databázový vývoj a generování dat.

Ultimate

Tato edice je nejvyšší. Rozšiřuje předešlou edici Premium. Obsahuje všechny potřebné nástroje k vývoji od návrhu po nasazení. Přináší užitečné nástroje modelování architektury a funkce pro spolupráci v týmu.

2.2 Visual Studio 2010 Express

Tato edice je oproti placeným hodně omezena. Nepodporuje zároveň všechny jazyky technologie .NET, proto je potřeba stáhnout verzi této edice podle potřeb vývojáře. Žádná z těchto verzí nepodporuje rozšíření, což je celkem velká nevýhoda, protože některé rozšíření jsou velice užitečné. Je to však od firmy Microsoft pochopitelné, protože rozšíření mohou přidat funkce které jsou až v placených edicích. Add-In co jsem vytvořil tedy nepůjde spustit na této edici.

Jelikož je tato edice zdarma i pro komerční použití, je vhodná do malých firem nebo pro samostatné vývojáře.

Windows Phone Developer Tools

Tato verze je uzpůsobena k tvorbě aplikaci na mobilní zařízení s Windows Phone. Hry je možné tvořit pomocí XNA frameworku. Pro aplikace je vhodnější použít technologii Silverlight. Je zde možné tvořit hry za použití obou technologií zároveň, kdy menu je tvořeno pomocí Silverlight a hra samotná se vykresluje pomocí XNA.

Visual Web Developer 2010 Express

Tato verze umožňuje vývoj webových aplikací pomocí technologií ASP.NET a JavaScript. S touto verzi se nainstaluje Web Server k testování webových aplikací a nástroje pro práci s databázemi.

Visual Basic 2010 Express

Tato verze umožňuje vývoj .NET aplikací pomocí jazyku Visual Basic.

Visual C# 2010 Express

Tato verze umožňuje vývoj .NET aplikací pomocí jazyku C#.

Visual C++ 2010 Express

Tato verze je vytvořena k tvorbě aplikací v jazyce C++. Je možné si vybrat zda chceme využívat technologii .NET nebo programovat nativní Windows aplikace, které jsou mnohem rychlejší.

2.3 Visual Studio 2012

Tyto edice se výrazně neliší od edic 2010, především jsou uzpůsobeny k vývoji pro Windows 8 a výrazně změnily vzhled.

2.4 Visual Studio 2012 Express

V nové řadě Visual Studia byly provedeny menší změny v této bezplatné edici. Především je zde verze Express pro klasický Windows Desktop, ale podporuje více programovacích jazyků zároveň jako jsou C#, Visual Basic, C++ atd. A potom je zde verze pro Windows 8, která umožňuje tvořit aplikace a hry do obchodu Windows Store, který je integrován ve Windows 8. Je zde i nově verze Visual Studio Team Foundation Server Express 2012, která je vhodná pro malé firmy, protože umožňuje spolupráci až pěti vývojářů v týmu.

3 Možnosti rozšíření Visual Studia

Pokud vývojářům nestačí funkce Visual Studia nebo způsob jakým je nějaká funkce ve Visual Studiu řešena, je možné si toto IDE rozšířit podle jejich představ. Buď vytvořením nějakého rozšíření přesně na míru nebo stažením již vytvořeného.

V této práci se zabývám tvorbou add-inů, jež umožňují rozšíření Visual Studia. Add-iny ale nejsou jedinou možností rozšíření Visual Studia. Visual Studio je možné dále rozšiřovat pomocí průvodců (Wizards), Visual Studio balíčků (VSPackages), MEF komponent a Maker.

K tvorbě add-inů a průvodců je potřeba použít nějakou placenou edici Visual Studia. Tvorba těchto rozšíření je již zabudována v těchto edicích a tak není potřeba nic doinstalovat. Naopak v edici Express není možné používat add-iny ani jiné druhy rozšíření.

Při vytváření balíčků (VSPackages) a MEF komponent samotné Visual Studio nestačí a je potřeba ještě doinstalovat Visual Studio 2010 SDK, které umožňuje vytvářet tato rozšíření. SDK obsahuje i různé ukázkové kódy a dokumentaci.

3.1 Add-iny

Jedná se o high-level rozšíření Visual Studia. Jsou to aplikace, které se dají integrovat do Visual Studia. Umožňují přidat nové funkce nebo zlepšit stávající. Add-in je COM objekt implementující rozhraní `IDTExtensibility2` a může komunikovat se samotným Visual Studiem přes automatizovaný objektový model. Add-in kompilovaný do DLL souboru, takže je tím chráněn vůči přístupu ke zdrojovým kódům a zároveň to zvyšuje výkon. K použití ani vytváření add-inů není potřeba mít nainstalované Visual Studio SDK.

3.2 Průvodci (Wizards)

Je to další druh rozšíření umožňující zjednodušit nějaký úkon. Průvodci jsou především tvořeny z několika oken, kterými uživatel prochází a vyplňuje nějaké hodnoty nebo vybírá určité možnosti, které jsou dále použity. Visual Studio využívá průvodce k vytvoření add-inu nebo založení nového projeku u Win32 konzolové aplikace. Každý průvodce musí implementovat `IDTWizard` rozhraní. K použití průvodců ani jejich vytváření není potřeba mít nainstalované Visual Studio SDK.

3.3 Balíčky (VSPackages)

Jedná se o low-level rozšíření, proto má ještě větší možnosti než add-iny. Tyto balíčky umožňují přidávat podporu nových programovacích jazyků do Visual Studia nebo vytvořit nový editor. Visual Studio je dokonce z velké části tvořeno těmito balíčky. Díky nižší úrovni programování jsou náročnější na tvorbu než add-iny. K použití a vytváření těchto rozšíření je už potřeba mít nainstalované Visual Studio SDK.

3.4 MEF Komponenty

Toto je zajímavá možnost rozšíření editoru Visual Studia 2010. MEF je knihovna k tvoření rozšířitelných aplikací. Je součástí .NET Frameworku 4. Je to určitá standardizace vytváření rozšíření nějaké aplikace.

MEF není výsadou Visual Studia, ale je možné tohoto frameworku využít i u vlastních aplikací třeba postavené na technologiích Windows Forms, ASP.NET a jiných. Editor Visual Studia je postaven na technologii WPF a je navržen k používání MEF komponent a sám je tvořen z těchto komponent.

MEF komponenta je určitá třída, která může využívat jinou komponentu a dokonce být použita jinou komponentou a to i zároveň. K použití těchto rozšíření je potřeba mít nainstalované Visual Studio SDK.

3.5 Makra

Jedná se o další typ rozšíření Visual Studia. Microsoft tento způsob využívá i u jeho jiných produktů jako jsou nástroje Word a Excel z Microsoft Office. Makra slouží především k automatizaci některých úkonů, které se stále opakují a pro programátora by byli zbytečně pracné.

Jako Add-iny využívají automatizovaný objektový model Visual Studia, ale nejsou integrovány do Visual Studia jako add-iny. Makra nemohou být vytvářena v Express edici Visual Studia. Nevýhodou maker může být, že uživatel který je používá vidí zdrojový kód.

Makra je možné programovat nebo používat Macro Recorder ve Visual Studiu 2010, který umí nahrát určité úkony co provede uživatel ve Visual Studiu a převádí je do kódu který je možný kdykoliv použít nebo upravit.

S verzí Visual Studia 2012 byly makra odstraněny, proto je nutné je převést na add-iny, které jsou jim podobné.

4 Možnosti nasazení Visual Studio rozšíření

Vytvořený add-in nebo jiné Visual Studio rozšíření je vhodné zaobalit do něčeho jako instalační průvodce, protože každý typ rozšíření se jinak vkládá do Visual Studia. Uživatel jež by chtěl dané rozšíření použít, by jinak byl nucen vědět, kam dané soubory vložit, popřípadě zařídit ještě jiné požadavky.

Proto Visual Studio nabízí tři formáty, jak nasadit do provozu tato rozšíření VSIX, MSI a VSI. Tyto formáty jsou takzvané kontejnery, které se postarají o instalaci určitého rozšíření bez nutnosti cokoliv vědět. Každý tento formát má určité vlastnosti a nemusí podporovat zrovna daný typ rozšíření.

Tabulka 1 znázorňuje tři základní druhy rozšíření a k nim podporované formáty k nasazení.

4.1 VSIX

Tento formát je možné nahrát do Visual Studio Galerie a tak umožňuje jednoduchou publikaci rozšíření. Samotný soubor s příponou .vsix může obsahovat i několik rozšíření a hlavně metadata, které využívá Extension Manager ve Visual Studiu. Umožňuje aktualizace, takže Visual Studio pokud vyjde nová verze, informuje uživatele o aktualizaci přes tento Extension Manager. Nepodporuje však zápis do GAC nebo systémových registrů. Nevýhoda pro add-iny je, že nejsou podporovány a je proto nutné použít jinou možnost.

4.2 MSI

Tento způsob je vhodný pro téměř všechny druhy rozšíření a jiných aplikací. Stačí přidat Setup Project do Solution s add-inem nebo jiným rozšířením a nastavit kam se má co nakopírovat. Po buildnutí Solution se vytvoří soubor s příponou .msi, který je pak možné publikovat i do Visual Studia Galerie s možností přístupu přes Visual Studio Extension Manager. Po spuštění .msi souboru je uživatel proveden jednoduchým průvodcem, který dané rozšíření nainstaluje. Dané rozšíření je potom možné i odinstalovat.

MSI podporuje i zápis do GAC nebo registrů. Nepodporuje ale aktualizace nebo aktivaci a dektivaci v Extension Manageru.

4.3 VSI

Tento formát je jednodušší k použití než MSI, ale nepodporuje jej Visual Studio Galerie. Dále nepodporuje zápis do GAC a systémových registrů a nemá možnost odinstalace. Pro add-in kterým se zabývám je ale vhodný, jen je potřeba počítat s tím, že musí být publikován na nějakých vlastních stránkách nebo šířit jinak(email, flash disk).

Je to komprimovaný soubor metodou ZIP s příponou .vsi, který obsahuje soubory daného rozšíření (u add-inů DLL soubor a registrační soubor s příponou .AddIn) a XML soubor s příponou.vscontent, který popisuje vložené soubory daného rozšíření. Více se o této možnosti zmíním v v části 8.6.2.

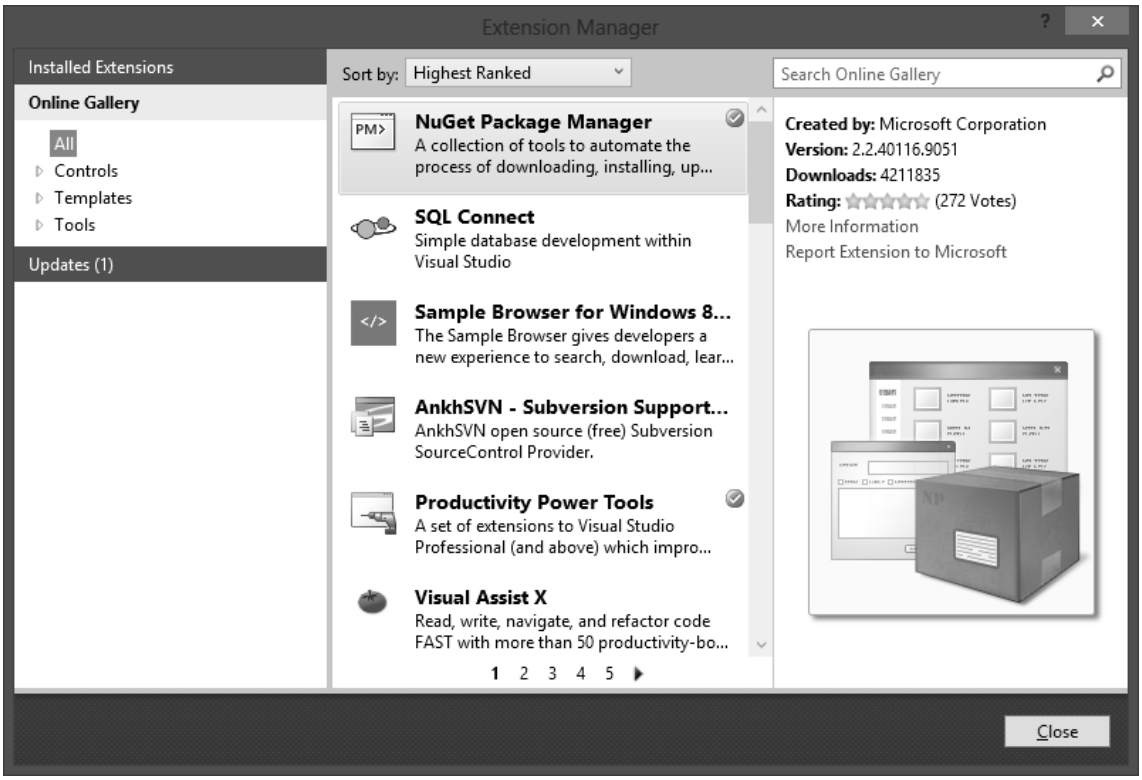
Typ rozšíření	**VSIX**	**MSI**	**VSI**
Add-in	Ne	Ano	Ano
Balíček(VSPackage)	Ano	Ano	Ne
MEF Komponenta	Ano	Ano	Ne

Tabulka 1: Základní typy rozšíření a k nim podporované formáty nasazení

5 Extension Manager

Pro rozšíření Visual Studia o nové funkce je nejlepší použít Extension manager. Extension manager je okno, které se nachází ve Visual Studiu 2010 pod záložkou `Tools`, viz. obrázek 1. V jeho levé části se nachází online galerie rozšíření nebo rozšíření co jsou již nainstalována a jejich updaty. Díky tomuto manageru je tedy jednoduché instalovat, vypínat nebo odinstalovávat a updatovat jednotlivá rozšíření. Jiná cesta je jít rovnou na stánky Visual Studio Galerie, ale tam se dá dostat i přes tento Extension manager, po kliknutí na více informací o daném rozšíření, protože z této galerie Extension Manager stahuje všechna rozšíření.

Typicky mají rozšíření přípony .vsix a .msi. Jsou to instalátory, které se postarají o nainstalování daného rozšíření. V tomto Extension Manageru je možné aktivovat nebo deaktivovat rozšíření, kromě těch, které byli nainstalovány pomocí MSI instalátoru. Rozšíření je možné získat i jinými způsoby, třeba z webových stránek firem, které dané rozšíření vytváří nebo od jiných programátorů. Většina rozšíření je zdarma, a tak není problém si rozšířit Visual Studio o užitečné funkce. Je nutné brát na vědomí, že některá rozšíření nemusí být tvořena na použitou verzi Visual Studio. Na stánkách Visual Studio Galerie je možné také add-iny nebo jiná rozšíření publikovat, o tom se zmíním později.



Obrázek 1: Extension manager

6 Publikace rozšíření

Vytvořené rozšíření se dá publikovat na vlastních webových stránkách, avšak lepším způsobem je nahrát rozšíření do Visual Studio Galerie, ve které se nacházejí všechna možná rozšíření s možností přidávat komentáře nebo tato rozšíření ohodnotit. To mi dává zpětnou reakci od uživatelů, která je nutná k případnému vylepšování rozšíření. Publikovat add-in do této galerie je jednoduché s použitím instalátoru MSI.

Postup publikace do Visual Studio Galerie:

1. Na stránkách Visual Studio Galerie vyberu možnost Upload. Která spustí webového průvodce nahraní nějakého rozšíření.
2. V prvním kroku jsem dotázán na typ rozšíření. Jsou zde čtyři možnosti Tool, Control, Project or Item Template a Storboard Shapes. Pro vytvořený add-in zabalený do .msi instalátoru vyberu možnost Tool(nástroj).
3. V dalším kroku je potřeba vybrat soubor instalátoru .msi z počítače pro přímé nahraní do Visual Studio Galerie nebo mít .msi soubor na vlastních stránkách a vybrat jen odkaz k němu. Pro mě je výhodnější nahrát soubor přímo do galerie první možností.
4. V posledním kroku se zadává titulek, pod kterým bude add-in zobrazen, dále verze, jazyk, do jaké kategorie náleží (např.: Bezpečnost, Web, Modelování atd.), zda se jedná o placené rozšíření nebo zdarma, nějaký popis a vybrat podporované verze Visual Studia. Na konci je nutné potvrdit podmínky a nakonec potvrdit publikaci rozšíření.

7 Add-in

V této kapitole se již zabývám pouze add-inami. Jak add-in vytvořit pomocí jednoduchého průvodce ve Visual Studiu. Jak zajistit, aby o něm Visual Studio vědělo a bylo jej schopné používat. Dále popíšu základní třídu add-inu Connect, která byla průvodcem vygenerována. A nakonec zmíním add-in manager ve Visual Studiu.

7.1 Vytvoření

Nejjednodušší cestou jak vytvořit Visual Studio add-in je pomocí průvodce. Ten se nachází ve Visual Studiu 2010 v hlavní nabídce `File` → `New` → `Project`. Ve vytvořeném okně `New Project` je nutné vybrat `Other Project Types` → `Extensibility` → `Visual Studio Add-in`. Spustí se průvodce vytvořením add-inu.

V průvodci je možné vybrat, jakým jazykem bych chtěl programovat. Na výběr jsou jazyky C#, Visual Basic, Visual C++/CLR a Visual C++/ATL. Já používám jazyk C#.

Dále vyberu Visual Studio 2010 jako prostředí, ve kterém vytvořený add-in poběží. V průvodci je dále vhodné nechat pro add-in vytvořit spouštěcí položku v záložce `Tools`. Je možné si i vybrat, aby se add-in spouštěl po startu Visual Studia, ale to nebudu potřebovat.

Na konci se může nechat vygenerovat i About okno s informacemi o programátorovi nebo společnosti a add-inu.

Po dokončení průvodce se vygeneruje Solution, ve kterém je jediná třída Connect a XML soubor pro registraci s příponou .AddIn.

7.2 Registrace

Registrace add-inu se provádí pomocí souboru s příponou .AddIn, jak je vidět z výpisu 1. Tento soubor používá jednoduchý a rozšířený jazyk XML. Soubor musí být umístěn ve složce, která je kontrolována na tyto soubory Visual Studiem.

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<Extensibility xmlns="http://schemas.microsoft.com/AutomationExtensibility">
  <HostApplication>
    <Name>Microsoft Visual Studio</Name>
    <Version>10.0</Version>
  </HostApplication>
  <Addin>
    <FriendlyName>mujAddin</FriendlyName>
    <Description>pokusny add-in</Description>
    <Assembly>MyAddin1.dll</Assembly>
    <FullClassName>MyAddin1.Connect</FullClassName>
    <LoadBehavior>0</LoadBehavior>
    <CommandPreload>1</CommandPreload>
    <CommandLineSafe>0</CommandLineSafe>
  </Addin>
</Extensibility>
```

Výpis 1: Registrace Add-inu pomocí XML

7.3 Instalace

Po vytvoření jednoduchého add-inu vzniknou dva soubory, které mají stejný název, ale jinou příponu.

První soubor má příponu .AddIn, který slouží k registraci add-inu a obsahuje cestu k DLL assembly add-inu. Tento soubor je nutné vložit do složky která je kontrolována Visual Studiem. To se dá nastavit vybráním v hlavní nabídce `Tools` → `Options` → `Environment` → `Add-In` → `Macro Security` → `Add the path`. Druhý soubor má příponu .dll, což je DLL assembly add-inu. Ten je nutné nakopírovat na místo, kam registrační soubor odkazuje pomocí tagu `Assembly`. Pokud je v tagu `Assembly` jen např. MyAddin1.dll, musí být DLL soubor na stejném místě jako registrační soubor.

7.4 Třída Connect

Add-in musí obsahovat třídu `Connect.cs`, která implementuje rozhraní `IDTExtensibility2`, `IDTCommandTarget`. Tato třída implementuje metody, které jsou volány Visual Studiem, jak je vidět v tabulkách 2 a 3. Nejdůležitější jsou metody `OnConnection` a `Exec`. Pokud jsem si v průvodci vytvořením add-inu nechal vytvořit tlačítko v hlavní nabídce Visual Studia pod záložkou `Tools`, tak v metodě `OnConnection` je kód, který toto provádí. A to tak že add-in vytvoří příkaz a přiřadí mu tlačítko v nabidce `Tools` pomocí rozhraní `Command` v namespace `EnvDTE`. Následně pokud je toto tláčítko kliknuto, provede se metoda `Exec` add-inu, ve které se pomocí podmínky reaguje na potřebný příkaz pomocí jeho jména. Můžeme reagovat i na více příkazů a tím pádem i tlačítek. Do metody `Exec` je pak vhodné psát kód, co má add-in dělat, jak je vidět z výpisu 2.

```
public void Exec(string commandName, vsCommandExecOption executeOption, ref object varIn,
    ref object varOut, ref bool handled)
{
    handled = false;
    if (executeOption == vsCommandExecOption.vsCommandExecOptionDoDefault)
    {
        if (commandName == "ProcessAddin.Connect.ProcessAddin")
        {
            Form1 f = new Form1(_applicationObject);
            f.Show();

            handled = true;
            return;
        }
    }
}
```

Výpis 2: Metoda Exec ve třídě Connect

7.5 Ladění Add-inů

Ladění add-inů je problémovější než ladění klasických aplikací. Vytvořený DLL add-inu nelze otevřít v aktuálním Visual Studiu a ladit jej zároveň, protože tento DLL soubor je

Metoda	Popis
OnConnection	Tato metoda je zavolána, když je add-in spouštěn ve Visual Studiu.
OnStartupComplete	Tato metoda je zavolána, po spuštění Visual Studia.
OnAddInsUpdate	Tato metoda je zavolána, když je add-in spouštěn nebo uvolněn z Visual Studia.
OnBeginShutdown	Tato metoda je zavolána, když se Visual Studio ukončuje.
OnDisconnection	Tato metoda je zavolána, když je add-in uvolněn z Visual Studia.

Tabulka 2: Metody rozhraní IDTExtensibility2

Metoda	Popis
Exec	Tato metoda je zavolána, pokud je spuštěn nějaký příkaz (instance implementující EnvDTE.Command).
QueryStatus	Tato metoda je zavolána, pokud aktualizována dostupnost příkazu (instance implementující EnvDTE.Command).

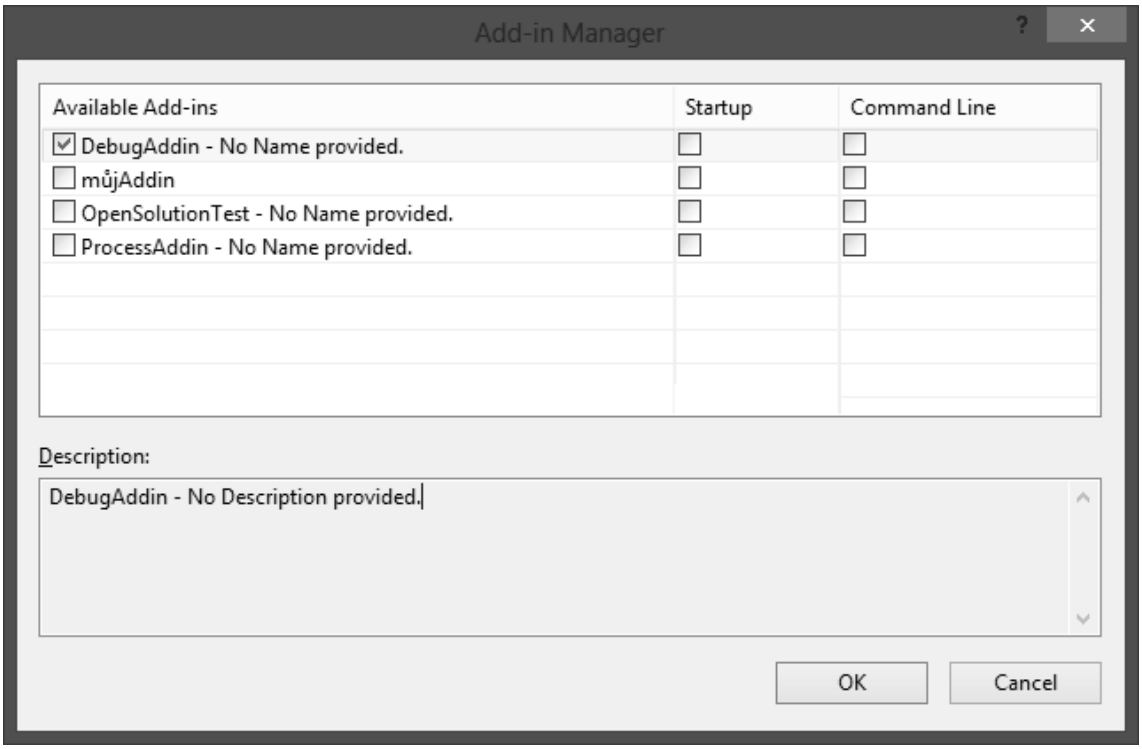
Tabulka 3: Metody rozhraní IDTCommandTarget

zamknut po načtení Visual Studiem. Proto je nutné vytvořit další instanci Visual Studia, která načte add-in a v té jej požívat. K ladění poslouží původní instance Visual Studia.

Kromě tohoto je ladění v původní instanci Visual Studia stejné jako jiných aplikací. Opět je možné do kódu přidávat třeba breakpointy a zjišťovat obsah proměnných atd.

7.6 Add-in Manager

Jedná se o nástroj Visual Studia, který umožňuje správu add-inů. Okno tohoto nástroje je vidět na obrázku 2. Obsahuje seznam registrovaných add-inů. Pokud jsou implementovány všechny metody třídy `Connect` add-inu správně, je možné pomocí checkboxů spustit nebo vypnout add-in. Dále je možné k add-inu nastavit, zda se má spustit při startu Visual Studia nebo třetí možnost je spustit add-in, pokud je Visual Stuio spouštěno přes příkazovou řádku.



Obrázek 2: Add-in manager

8 Add-in k ladění paralelních aplikací

V této kapitole se zabývám tvorbou add-inu, který zjednoduší ladění paralelních aplikací. První se zmíním o balíku nástrojů HPC Pack SDK, který je nutný k tvorbě paralelních aplikací. A dále řeším základní kroky vedoucí ke spuštění programu paralelně a attachnutí jeho procesů k Debuggeru Visual Studia. Ke konci tyto kroky použiji k návrhu add-inu.

8.1 HPC Pack 2008 SDK

Balík nástrojů sloužící k tvorbě paralelních aplikací pro super počítače postavené na platformě Windows HPC. V samotném add-inu používám zejména příkaz mpiexec, který umožňuje emulaci HPC Clusteru a tím spustit více procesů jednoho programu na jediném PC.

8.2 Základní kroky Add-inu

8.2.1 Otevření Solution laděného programu

Před samotným attachnutím spuštěných procesů, je nutné mít otevřené Solution programu, který chci ladit. Ve vytvořeném add-inu je instance `_applicationObject`, která implementovala rozhraní DTE2. Tato instance má vlastnost `Solution`, která vrací objekt implementující rozhraní `EnvDTE.Solution`. Díky tomuto objektu se dá přistupovat k otevřeným projektům právě otevřeného Visual Studia. V add-inu je potřeba zavřít aktuální práci, pokud je nějaká, a pak otevřít Solution potřebného programu k ladění, podobně jak je ve výpisu 3.

```
public void LoadSolution(String name)
{
    _applicationObject.Solution.Close();
    _applicationObject.Solution.Open(@"C:\Users\Jiri\Documents\Visual Studio 2010\Projects\"+
        name+@"\"+name+".sln");
}
```

Výpis 3: Metoda k otevření Solution

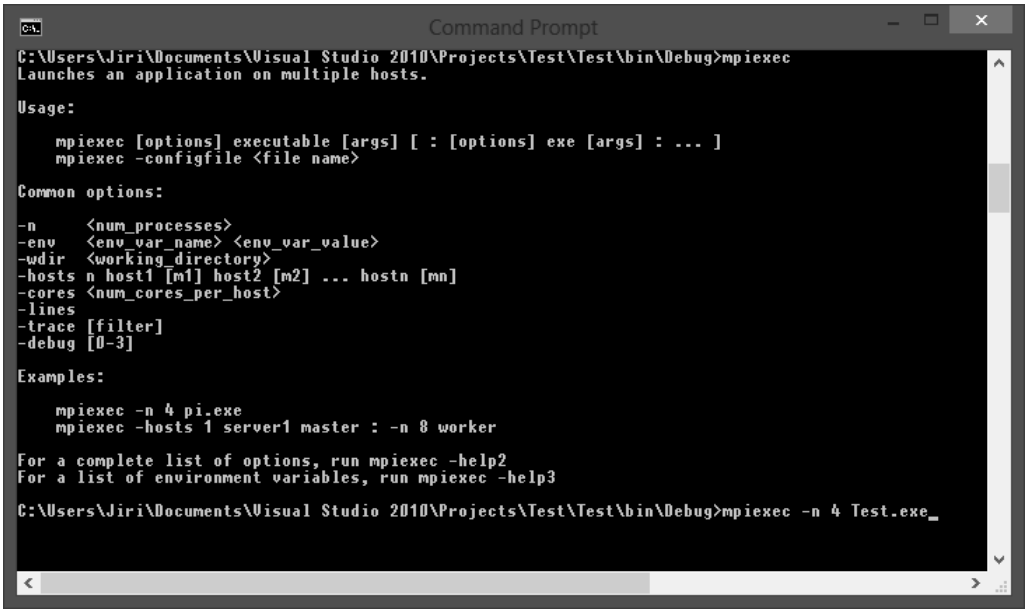
8.2.2 Spuštění programu několikrát

Pro spuštění programu vícekrát, využiju příkaz `mpiexec`, který můžu používat po nainstalováni balíku nástrojů HPC PACK 2008 SDK. Na obrázku 4 je vidět nápověda jak tento příkaz použít.

Pro ukázku tohoto příkazu jsem vytvořil jednoduchý program Test.exe, který každých pět sekund vypíše na konzoli "`Výpis n, procesu p`", kde `n` je číslo začínající od 0 a `p` je ID procesu tohoto programu. Jak je vidět na obrázku 3.

Příkazem `mpiexec -n 4 Test.exe` se tedy spustí jednoduchý program Test.exe 4x, každý vzniklý proces vypisuje na stejnou konzoli, jak je vidět na obrázku 5.

Obrázek 3: Výpis z programu Test.exe



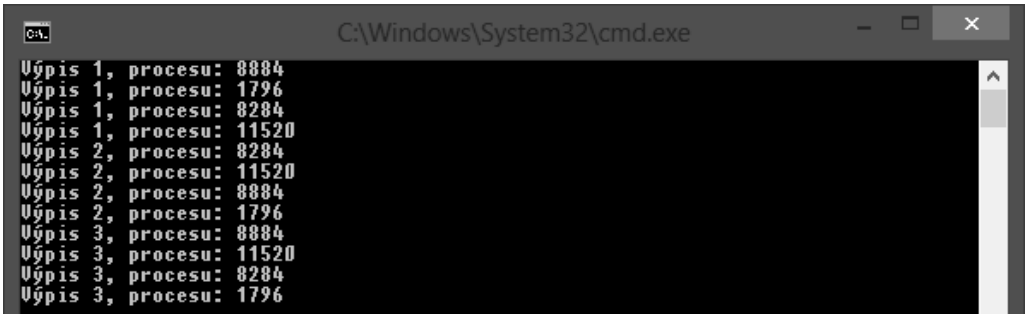
Obrázek 4: Nápověda příkazu mpiexec

Tohoto jsem využil v add-inu. Add-in může provádět příkazy pro konzoli, ve které se provede příkaz `mpiexec` a spustí tak požadovaný program. To lze provést pomocí třídy `Process` v namespace `System.Diagnostics`. Metoda ve výpisu 4 ukazuje, jak využít tuto třídu ke spuštění příkazového řádku. Pomocí `String` parametru této metody je možné provést v příkazovém řádku jakýkoliv podporovaný příkaz.

```
private Process CommandProcess { get; set; }
public void StartCommand(String command)
{
    CommandProcess = new Process();
    CommandProcess.StartInfo.FileName = "cmd.exe";
    CommandProcess.StartInfo.Arguments = "/C␣" + command;
    CommandProcess.Start();
}
```

Výpis 4: Ukázka použití třídy Process z namespace System.Diagnostics

Pokud získám pole procesů běžících před voláním metody `StartCommand` ve výpisu 4 a po jejím volání, liší se tím, že v druhém poli procesů se nachází navíc pro-



Obrázek 5: Spuštění programu pomocí příkazu mpiexec

ces cmd.exe. Získat běžící procesy můžu opět pomocí třídy `Process` v namespace `System.Diagnostics` jak je vidět ve výpisu 5. Je zde ale problém, že bychom nezískali procesy programu, který bychom chtěli ladit. Po zapnutí příkazové řádky by došlo ihned k provádění dalšího příkazu v kódu, přitom je nutné aby se provedl příkaz `mpiexec` a spustil laděný program.

Proto je potřeba v metodě nějakou dobu čekat, než se příkaz v příkazové řádce provede. Jiný způsob než čekání určitou dobu není možný, proto používám třídu `Thread` a její metodu `Sleep`.

Dobu čekání ve výsledném add-inu nechám volitelnou, protože se může měnit podle výkonu počítače. Na rychlejším počítači s dvoujádrovým procesorem Intel Pentium E6300 stačilo bezpečně čekat 100ms. U notebooku s procesorem Intel Pentium P6200 ale při této hodnotě se ne vždy spustili všechny procesy, a proto jsem musel nastavit čekání na přibližně 300ms.

```
int Delay = 300;
Process[] processesBeforeStart = Process.GetProcesses();
StartCommand(command);
Thread.Sleep(Delay);
Process[] processesAfterStart = Process.GetProcesses();
```

Výpis 5: Získání procesů pomocí třídy Process z namespace System.Diagnostics

Dvě vzniklá pole z výpisu 5 je potřeba nějakým způsobem zanalyzovat. Pokud byl totiž úspěšně proveden příkaz `mpiexec`, tak ve druhém poli bude více procesů. Zkoumáním druhého pole jsem zjistil, že kromě procesů programu k ladění vznikly i jiné procesy. Mezi jiné procesy patří `cmd`, `smpd`, `conhost`, `mpiexec` a `WerFault`.

Vytvořil jsem tedy metodu z výpisu 6, která vrátí ID procesů programu k ladění a zbylé procesy uloží do generické kolekce `List`. Zbylé procesy ukládám, protože se nemusí všechny ukončit při konci ladění nebo když dojde k nějaké chybě, proto je nakonec zavírám pomocí nějaké metody, pokud jsou ještě aktivní.

```
private List<Process> OtherProcesses { get; set; }

private List<int> GetNewProcesses(Process[] processesBeforeStart, Process[]
    processesAfterStart, String nameOfProcessToAttach)
```

```
{
    bool save = false;
    List<int> processes = new List<int>();

    foreach (Process procAfter in processesAfterStart)  // if procAfter is in processesBeforeStart
        than do not save
    {
        save = true;
        foreach (Process procBefore in processesBeforeStart)
        {
            if (procBefore.Id == procAfter.Id)
            {
                save = false;
                break;
            }
        }

        if (save == true && procAfter.ProcessName == nameOfProcessToAttach)
        {
            processes.Add(procAfter.Id);
        }
        if (save == true)
        {
            OtherProcesses.Add(procAfter);
        }
    }

    return processes;
}
```

Výpis 6: Metoda k získání potřebných procesů

8.2.3 Attachnutí požadovaných procesů

Zavoláním metody GetNewProcesses z výpisu 6 získám potřebné ID procesů k attachnutí. Abych tyto procesy mohl attachnout, je ale potřeba získat instance procesů třídy `Process`, ale v namespace `EnvDTE`.

K získání procesu z namespace `EnvDTE` slouží metoda GetProcessById a k attachnutí tohoto procesu slouží metoda AttachProcess, jak je vidět z výpisu 7.

```
private DTE2 _applicationObject;

public void AttachProcess(int id)
{
    EnvDTE.Process process;
    process = GetProcessById(id);

    process.Attach();
}


private EnvDTE.Process GetProcessById(int id)
```

```
{
    EnvDTE.Process process = null;
    EnvDTE.Processes processes = _applicationObject.Debugger.LocalProcesses;

    foreach (EnvDTE.Process p in processes)
    {
        if (p.ProcessID == id)
        {
            process = p;
            break;
        }
    }
    return process;
}
```

Výpis 7: Získání procesu z namespace EnvDTE a jeho Attachnuti

8.2.4 Přejít do režimu ladění

Jako při ladění klasických programů, by bylo dobré mít možnost přidat breakpoint kdekoliv v programu. V tomto je ale problém. Add-in při spuštění příkazu mpiexec nečeká, než tento příkaz spustí procesy programu k ladění. Tím pádem ještě než jsou procesy vytvořeny add-in vykonává další příkazy a nebylo by možné zjistit tyto nové procesy. Proto jsem přidal mezi spuštěním a zjištěním nových procesů určitou odezvu. Tím se vyřeší jeden problém, ale přijde další.

Zjištěné procesy se attachnou k debuggeru, ale než jsou attachnuty, tak program už nějakou chvíli běží. Díky tomu pokud přidám breakpoint na začátek Main metody třeba automaticky pomocí příkazu z výpisu 8, tak tento brekapoint nemůže být hitnut.

Proto je nutné použít složitější způsob. A to třeba přidat na začátek Main metody určitou odezvu, která stačí stejná jako v add-inu. A potom přidat breakpoint, ten by ale musel být přidán až na další příkaz v programu k ladění. Zjišťovat další příkaz se mi zdálo složité, a tak při přidání odezvy přidávám i break kód.

Touto cestou je nutné už najít Main metodu. Program k ladění může obsahovat více tříd, a tak je nutné procházet všechny třídy v projektech a hledat tuto Main metodu. Jak je vidět ve výpisu 9. Po získání objektu Main metody implementující rozhraní CodeElement z namespace EnvDTE, přidávám odezvu s break kódem na její začátek, viz. výpis 10. Break kód přidávám díky třídě `Debugger` z namespace `System.Diagnostics`.

```
applicationObject.Debugger.Breakpoints.Add("Main");
```

Výpis 8: Možnost jak přidat Breakpont programově

```
private void LocateMain()
{
    foreach (Project p in _applicationObject.Solution.Projects)
    {
        foreach (ProjectItem projectItem in p.ProjectItems)
        {
            String name = projectItem.Name;
```

```
            FileCodeModel fileCodeModel = projectItem.FileCodeModel;

            if (fileCodeModel != null)
            {
                CodeElements codeElements = fileCodeModel.CodeElements;

                Hierarchy(codeElements);
            }
        }
    }
}

private void Hierarchy(CodeElements elems)
{
    foreach (CodeElement elem in elems)
    {
        try
        {
            if (elem.Name == "Main")
            {
                main = elem;
                break;
            }
            if (elem.Children != null)
                Hierarchy(elem.Children);
        }
        catch
        {
            // nekdy neni elem.Children implementovan, neni treba reagovat
        }
    }
}
```

Výpis 9: Metody k nalezení Main metody

```
private void AddSpecificCode()
{
    TextPoint textPoint = main.GetStartPoint(vsCMPart.vsCMPartBody);
    EditPoint editPoint = textPoint.CreateEditPoint();

    String text = editPoint.GetLines(editPoint.Line, editPoint.Line + 2);
    text = text.Replace(" ", "");

    if (text != "System.Threading.Thread.Sleep(" + delay + ");\r\nSystem.Diagnostics.Debugger.
        Break();")
    {
        editPoint.Insert("System.Threading.Thread.Sleep(" + delay + ");\r\n");
        lineToBreak = editPoint.Line;
        editPoint.Insert("System.Diagnostics.Debugger.Break();\r\n");
    }
    else
    {
        throw new BreakCodeAlreadyExistsException("Break code already exists!");
    }
```

```
    }
```

Výpis 10: Metoda k přídání break kódu

8.2.5 Nastavit Visual Studio pro ladění

Protože add-iny mají přístup k objektu DTE2, můžu automaticky nastavovat toto prostředí nebo otevírat okna, než abych to řešil ručně. Při ladění paralelních aplikací se hodí mít otevřené okno `Processes`, kde je možné vidět attachnuté procesy. Dále je dobré mít otevřené okna `Threads` nebo `Breakpoints`. Oproti jiným oknům okna `Processes` a `Threads` nejdou otevřít vždy, ale pouze při ladění. Toto je nutné vědět při programování add-inu. K získání instance okna slouží kolekce `Windows` objektu DTE nebo DTE2. Pokud budu chtít otevřít nějaké známé okno, použiji metodu `Item` této kolekce a do parametru dám konstantu třídy `EnvDTE.Constants`, ktetá obsahuje GUID těchto oken např.: `EnvDTE.Constants.vsWindowKindToolbox`.

Kolekce `Windows` obsahuje aktuálně spuštěná okna, ale metodou `Item` se dá nejen získat instance spuštěných oken, ale i získat ostatní neotevřená okna. Pokud mezi konstantami není okno co potřebujeme, musíme získat jeho GUID jinak. K tomu jsem si udělal jednoduchý add-in, který prochází tuto kolekci a vypíše vlastnosti `Caption`, titulek a `ObjectKind`, což je GUID okna. Stačí pak ve Visual Studiu spustit nějaké okno a pomocí add-inu zjistit jaké má GUID. Pro zjednodušení jsem vybral GUID některých oken, jak je vidět v tabulce 4. Ke spouštění těchto oken je možné použít metodu SetForDebugging z výpisu 11.

Titulek	GUID
Threads	{E62CE6A0-B439-11D0-A79D-00A0C9110051}
Processes	{51C76317-9037-4CF2-A20A-6206FD30B4A1}
Start Page	{387CB18D-6153-4156-9257-9AC3F9207BBE}
Toolbox	{B1E99781-AB81-11D0-B683-00AA00A3EE26}
Object Browser	{269A02DC-6AF8-11D3-BDC4-00C04F688E50}
Error List	{D78612C7-9962-4B83-95D9-268046DAD23A}
Solution Explorer	{3AE79031-E1BC-11D0-8F78-00A0C9110057}

Tabulka 4: Vybraná okna Visual Studia a k nim přiřazených GUID

Visual Studio	Obsah tagu Version
Visual Studio 2008	**9.0**
Visual Studio 2010	**10.0**
Visual Studio 2012 Page	**11.0**
Jakákoliv verze	*

Tabulka 5: Závislost tagu Version registračního souboru na verzi Visual Studia

```
public void SetForDebugging()
{
    //Threads, {E62CE6A0−B439−11D0−A79D−00A0C9110051}
    //Processes, {51C76317−9037−4CF2−A20A−6206FD30B4A1}
    Window w1 = _applicationObject.Windows.Item("{51C76317−9037−4CF2−A20A−6206
        FD30B4A1}");
    w1.Visible = true;
    Window w2 = _applicationObject.Windows.Item("{E62CE6A0−B439−11D0−A79D−00
        A0C9110051}");
    w2.Visible = true;
}
```

Výpis 11: Metoda nastavující Visual Studio k ladění

8.3 Podpora pro Visual Studio 2010 i 2012

Vytvořený add-in k ladění paralelních aplikací by měl splňovat další požadavky. V průběhu tvorby této bakalářské práce byla uvolněna finální verze Visual Studia 2012. Proto aby add-in nebyl vyvíjen jen na starší verzi Visual Studia, tak bylo vhodné, aby fungoval i na nové verzi Visual Studia 2012. Pro podporu ve Visual Studiu 2012 je potřeba pouze upravit registrační soubor DebugAddin.Addin ve výpisu 12 a to tak, že se změní verze v tagu Version z 10.0 na 11.0. Obsah tagu je závislá na verzi Visual Studia, vice je v tabulce 5

```
<HostApplication>
    <Name>Microsoft Visual Studio</Name>
```

```
    <Version>11.0</Version>
</HostApplication>
```

Výpis 12: Úprava registračního souboru pro podporu Visual Studia 2012

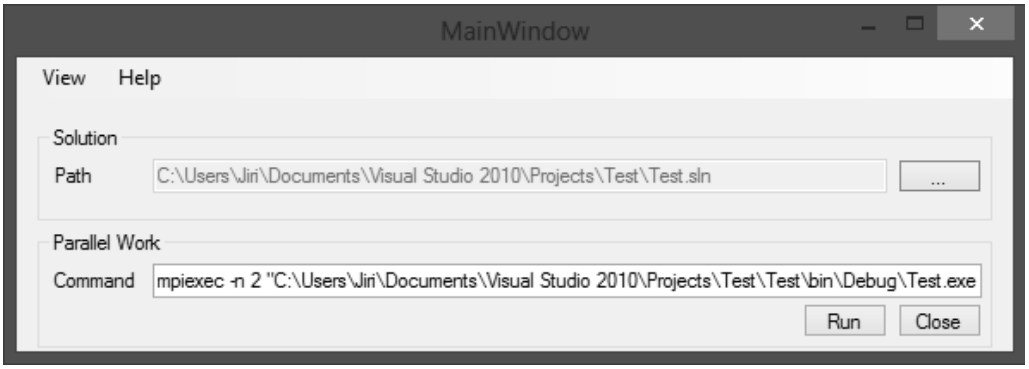
8.4 GUI Add-inu

K základní funkci add-inu slouží jednoduché GUI, viz. obrázek 6, které umí otevřít Solution programu k ladění a dále Textbox, do kterého se vygeneruje základní mpiexec příkaz. Tlačítko Run spustí příkazem proces a attachne jej k debuggeru. Pokud nebyl přidán Break kód na začátek programu, tak jej add-in přidá. Tlačítkem Close se zavřou všechny vzniklé procesy a ukončí se ladění. To samé se provede před zavřením add-inu. Menu tohoto okna umožní zobrazit okno "O aplikaci". A dále umožní zobrazit rozšířenou verzi hlavního okna, jak je vidět na obrázku 7. V rozšířené verzi se dají nastavit další věci jako upravovat odezvu mezi spuštěním příkazu mpiexec a attachnutím některých vzniklých procesů nebo vidět ostatní spuštěné procesy po tlačítku Run.

8.5 Návrh Add-inu

Jednotlivé třídy add-inu jsem navrhl tak, aby se každá třída zabývala určitou problematikou. Metody těchto tříd stačí potom volat v určitém pořadí. K tomuto využívám návrhový vzor Facade (Fasáda).

- **CommandLine** - Tato třída pracuje s příkazovou řádkou a provádí příkazy pomocí metody StartCommand s příkazem ve String parametru.
- **VisualStudioCustomizer** - Tato třída umožní spustit některá okna Visual Studia, třeba okno Processes.
- **Analyzer** - V této třídě jsou metody BeginAnalyze a EndAnalyze, které se používají před a po volání nějakého příkazu, dále zde je metoda, která vrátí procesy k attachnutí a property, která ukládá ostatní vytvořené procesy.
- **Attacher** - Tato třída umí attachnout nějaký proces podle jeho ID nebo jej detachnout.
- **Builder** - Pomocí této třídy je možné buildnout aktuální Solution.
- **Solution** - Tato třída má metody k otevření Solution podle cesty a vracet cestu k souboru ve složce Debug tohoto Solution.
- **ICode** - Toto rozhraní má metody k přidávání a odebírání určitého break kódu. Toto rozhraní jsem použil kvůli tomu, aby bylo možné add-in snadno rozšířit pro podporu ladění aplikací napsaných i v jiných jazycích.
- **CsCode** - Tato třída implementuje rozhraní ICode a umožňuje tak přidávat break kód pro jazyk C#.



Obrázek 6: GUI Add-inu

- **ICommand** - Toto rozhraní má metodu k vytvoření příkazu pro příkazovou řádku se String parametrem, který předá cestu k tomuto souboru. Implementací tohoto rozhraní je možné vracet různé příkazy.
- **HPCPackCommand** - Tato třída implementuje ICommand rozhraní a metodou GetCommand vrátí příkaz mpiexec ke spuštění souboru paralelně.

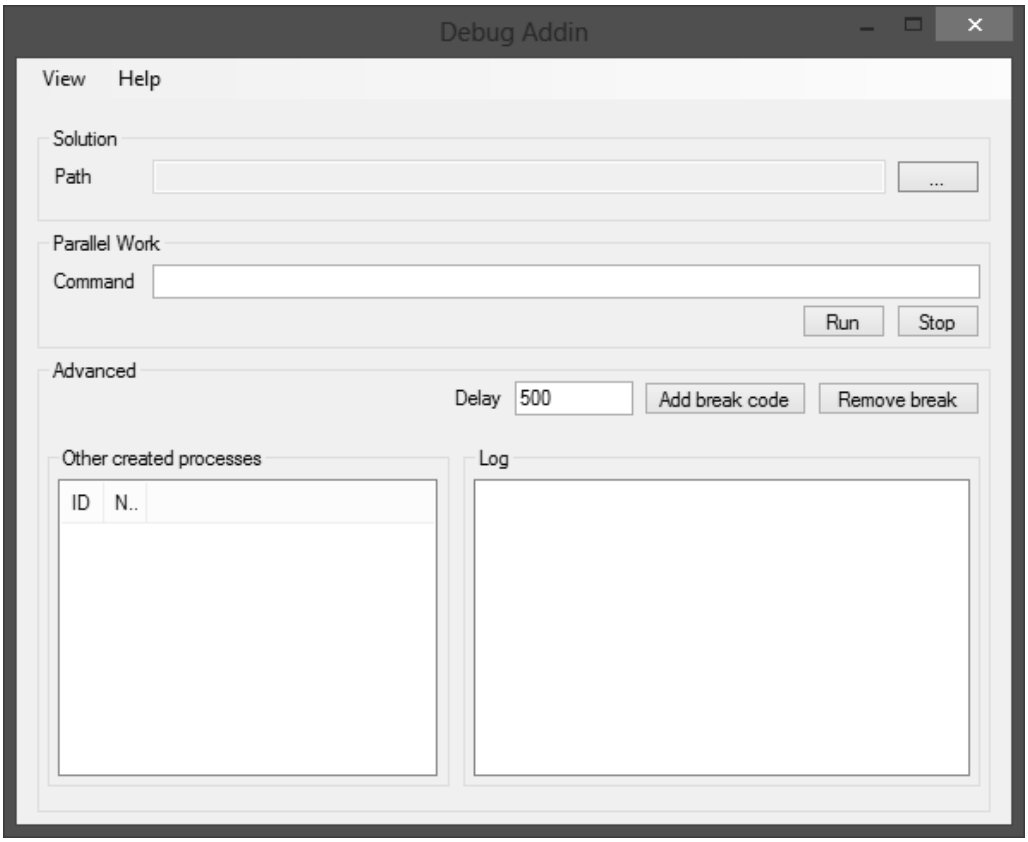
Všechny třídy podle vzoru Facade používá jedna třída, kterou jsem pojmenoval jako vzor Facade. Nejdůležitější metody jsou Start, která přidá break kód, spustí program paralelně a přepne do režimu ladění a metoda End, která ladění ukončí a odebere break kód. Pro přehlednost jsem implementoval touto třídou rozhraní IFacade které má hlavičky všech public metod a vlastností.

Třída Facade nevytváří objekty třídy CsCode ani HPCPackCommand, ale přebírá tyto objekty pomocí konstruktoru. K těmto třídám přistupuje pomocí jejich rozhraních, tím je možné použít třídy s jinou implementací.

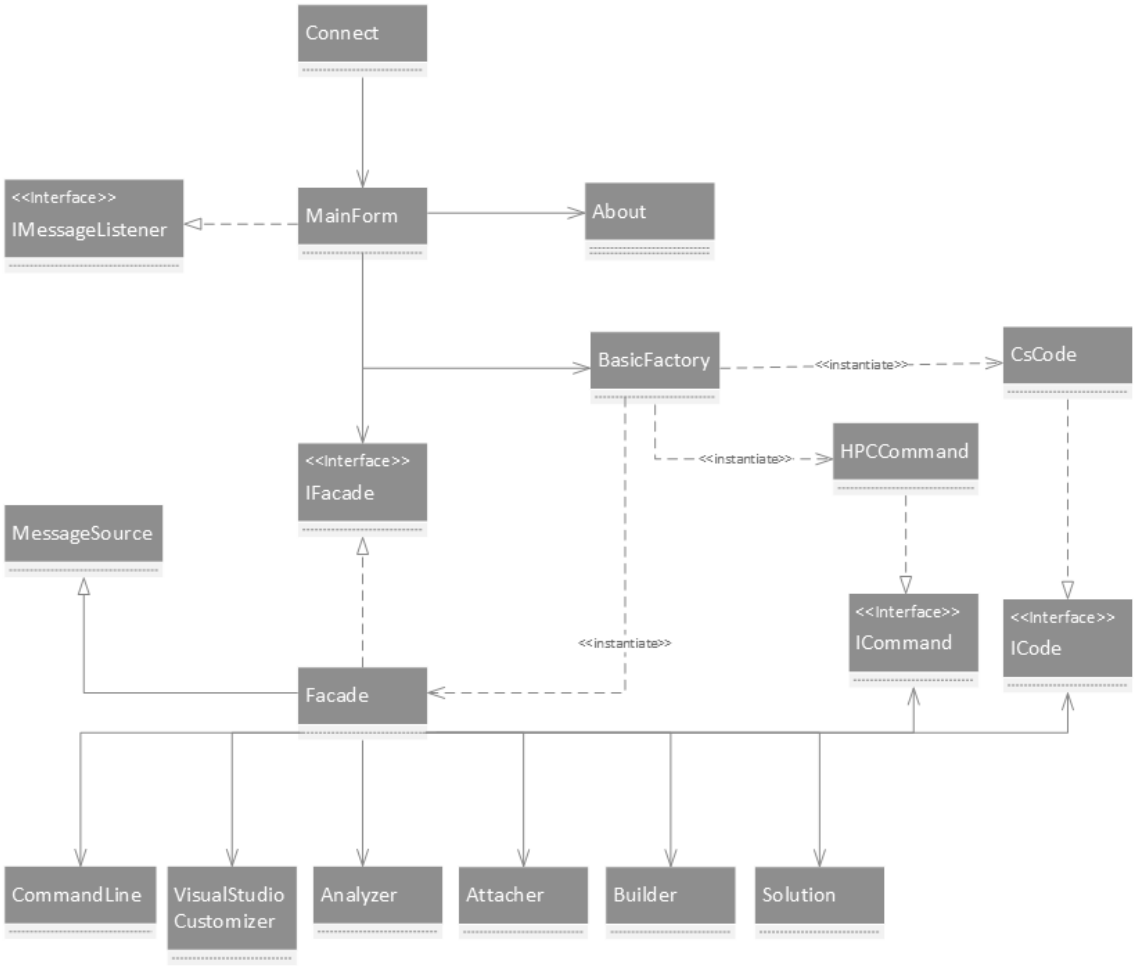
K vytvoření požadovaných tříd používám zjednodušený vzor Factory, kdy třída BasicFactory metodou GetFacade vrátí objekt Facade, který používá požadovaný programovací jazyk a příkaz ke spuštění programu paralelně. Ve třídě Connect tedy zobrazím Windows Forms formulář, který pomocí BasicFactory získá požadovanou Factory a tu použije.

Dále jsem chtěl mít ve formuláři možnost vidět nějaký log proto jsem navrhl třídu MessageSource a rozhraní IMessageListener. Formulář implementuje rozhraní IMessageListener a jeho metodu PrintLog, která slouží k vypsání logu. Objekt Facade rozšiřuje třídu MessageSource a slouží jako zdroj zpráv a volá pomocí delegáta metodu posluchače. Tento způsob je zjednodušený oproti klasickým událostem, kdy se dá použít ještě třída EventArgs k posílání více parametrů než jedné zprávy.

Ve zjednodušeném UML třídním diagramu na obrázku 8 nemám zapsány třídy vlastních výjimek, protože by zabíraly zbytečně místo, to samé nejdou vidět v UML třídním diagramu metody a vlastnosti. Především ukazuji jak spolu třídy souvisí.



Obrázek 7: Rozšířené GUI Add-inu



Obrázek 8: Zjednodušený UML třídní diagram add-inu

8.6 Instalační průvodce

V této části popíšu, jak nasadit add-in pomocí kontejnerů MSI a VSI, jež podporují add-iny. Základní informace o těchto možnostech nasazení již bylo zmíněno v kapitole 4.

8.6.1 MSI

Tato metoda je asi, nejideálnější pro add-iny, protože je můžu potom publikovat ve Visual Studio Galerii, což mi dává větší šanci add-in rozšířit. Jak bylo napsané v kapitole 4, tato metoda vytvoří instalačního průvodce, který už nakopíruje soubory add-inu na požadované místo.

Postup je následující:

1. Do Solution vytvořeného add-in přidám projekt Setup Project. Tento projekt se nachází v `Other Project Types` → `Setup and Deployment` → `Visual Studio Installer`.

2. Dále je dobré přidat všechny závislosti DLL assembly add-inu. K zjištění všech těchto závislostí je nutné klinout pravým tlačítkem myši na vytvořený Setup Project a dát `Add` → `Project Output`. Ve vytvořeném okně vybrat Primary Output pro add-in projekt a kliknout OK. Zajistí to, aby instalační soubor nakopíroval DLL assembly add-inu a jiné soubory, na kterých je tento add-in závislý na nějaké místo. Chybí zde ale registrační soubor add-inu.

3. Po klinutí opět pravým na Setup Project, vybereu `View` → `File System`. Toto otevře okno ve kterém je znázorněn Cílový počítač a k němu tři složky. Ve složce Application Folder, což je složka typicky pro aplikaci, jsou vloženy naleznuté závislosti add-inu a DLL soubor add-inu jako výstup projektu add-inu. K těmto souborům je nutné přidat registrační soubor kliknutím pravého tlačítka myši na tuto složku pak `Add` → `File`. Přidat DLL je možné i přes `Add` → `Assembly`. Tímto přidáním pokud se neprovedl krok 2 jsou zjištěny z DLL assembly add-inu závislosti a automaticky přidány, ale tento způsob nemusí najít všechny závislosti.

4. Následně chci aby add-in byl přidán do určité složky a to Dokumentů, kde má Visual Studio svoji složku a v ní složku Addins do které je nutné nakopírovat soubory add-inu. Tato cesta se dá zapsat jako `[PersonalFolder]\Visual Studio 2010\Addins`. Tato cesta se zadá ve vlastnostech Application Folder v property DefaultLocation.

5. Předposledním krokem by mělo být přidat určité informace k instalátoru add-inu, jako jsou autor, popis, lokalizace, verze atd. To je možné přes vlastnosti Setup projektu.

6. Nakonec stačí jen buildnout Setup Projekt a ve složce bin tohoto projektu se budou nacházet 2 instalační soubory, jeden s příponou .exe a druhý s příponou .msi. Soubor s příponou .msi je instalátor, který se dá publikovat.

Výhoda tohoto způsobu je, že se dá add-in i se všema souborama, na kterých je add-in závislý a nemuseli by být v cílovém počítači, odinstalovat. Buď klasicky přes ovládací panely nebo pomocí instalačního souboru.

Pro podporu ve Visual Studio 2012 je nutné udělat druhý tento instalátor s upraveným registračním souborem add-inu nebo v registračním souboru nastavit u verze pro jaké Visual Studio je tento add-in určen, dát znak hvězdičky (*). Potom nechat zvolit uživatele v průvodci kam se soubory nakopírují, to je ale možné už teď.

8.6.2 VSI

Tato metoda je jednodušší než MSI, ale má určité omezení, popsané v kapitole 4. Dále jsem zjistil, že pokud máme nainstalované Visual Studio 2010 i 2012, tak se add-in nakopíruje do obou, ale díky registračnímu souboru add-inu, nelze úspěšně spustit add-in v obou vývojových nástrojích. Možnost jak toto řešit je, že v registračním souboru nespecifikuji verzi Visual Studia a dám tam *, tím pojede add-in v obou prostředích.

Postup je následující:

1. Vytvořený add-in obsahuje soubory, v mém případě DebugAddin.dll a DebugAddin.AddIn. K těmto souborům přiřadíme XML soubor DebugAddin.vscontent obsahující kód z výpisu 13. V tomto souboru je důležité mít nastaven element ContentVersion na 2.0 kvůli podpoře tohoto instalátoru pro Visual Studiu 2008, 2010 a 2012.

2. Tyto tři soubory zkomprimuju metodou ZIP a výsledný DebugAddin.zip soubor přejmenuju na DebugAddin.vsi. Windows po přejmenování na .vsi rozpozná tento typ a změní ikonku tohoto souboru.

3. Tento VSI soubor je pak možné publikovat. Po spuštění tohoto souboru se spustí instalátor, který soubory add-inu nakopíruje do

   `n:\...\My Documents\Visual Studio <version>\Addins\.`

```
<VSContent xmlns="http://schemas.microsoft.com/developer/vscontent/2005">
    <Content>
        <FileName>DebugAddin.Addin</FileName>
        <FileName>DebugAddin.dll</FileName>
        <DisplayName>Debug Add-in</DisplayName>
        <Description>Add-in to easier debugging parallel applications.</Description>
        <FileContentType>Addin</FileContentType>
        <ContentVersion>2.0</ContentVersion>
    </Content>
</VSContent>
```

Výpis 13: Obsah souboru s příponou .vscontent v jazyku XML

9 Závěr

V této práci jsem se blíže seznámil s jednotlivými verzemi Visual Studia, to bylo nutné, protože add-in který jsem vytvořil, nelze použít v edicích Express. Dále add-iny obecně nelze používat na starší verze Visual Studia, než na jakou jsou vytvořeny. Add-in jsem vytvářel pro verzi 2010, tento add-in lze ale používat i na novější verzi Visual Studia 2012, která add-iny oproti makrům podporuje. Bylo jen nutné pamatovat na to, že je potřeba mít správně nastaven tag Version v registračním souboru add-inu.

Dále jsem si touto prací prohloubil znalosti jazyka C#, jelikož jsem add-in vytvářel v tomto jazyce. Potřeboval jsem pracovat s procesy z namespace System.Diagnostics a také z namespace EnvDTE. Také jsem musel pochopit automatizovaný objektový model Visual Studia, díky kterému je možné s Visual Studiem pracovat pomocí add-inů.

K vytvoření add-inu pro ladění paralelních aplikací, jsem se naučil použít HPC Pack 2008 ke spuštění programu paralelně příkazem mpiexec a to v add-inu využít. Samotné GUI je jednoduché, možná by se dalo v budoucnu ještě nějak zjednodušit nebo zabudovat více do Visual Studia.

Zopakoval jsem si také návrhové vzory, kdy ve výsledném add-inu bylo ideální použít návrhový vzor Facade (fasáda) v kombinaci se zjednodušenou verzí návrhového vzoru Factory (továrna). Tím jsem docílil i možnosti budoucího rozšíření, kdy je do add-inu možné přidat podporu pro ladění aplikací napsaných v jiných jazycích než C#.

Dále jsem se zabýval nasazením a publikací add-inu. Uživatele není vhodné nutit k ruční instalaci, a tak jsem použil jako možnost nasazení instalátory VSI a MSI. MSI má výhodu v tom, že zjišťuje závislosti add-inu na knihovnách. Pokud nějaké závislosti najde jsou tyto knihovny přenášeny s add-inem v tomto instalátoru. Oba instalátory umožňují nainstalovat add-in jak do verze Visual Studia 2010, tak do verze 2012. K publikaci add-inu i jiných rozšíření jsem popsal jak použít Visual Studio Galerii.

10 Reference

[1] Nayyeri Keyvan, *Professional Visual Studio Extensibility*, Wiley Publishing, Inc., 2008
ISBN: 978-0470230848

[2] Powers Lars, Snell Mike, *Microsoft Visual Studio 2010 Unleashed*, Sams, 2010
ISBN: 978-0672330810

[3] Bruce Johnson, *Professional Visual Studio 2012*, Wrox, 2012
ISBN: 978-1118337707

[4] Bruce Johnson, *Professional Visual Studio 2010*, Wrox, 2010
ISBN: 978-0470548653

[5] Alex Mackey, *Introducing .NET 4.0: With Visual Studio 2010*, Apress, 2010
ISBN: 978-1430224556

[6] Troelsen Andrew, *Pro C# 2010 and the .NET 4 Platform, Fifth Edition*, Apress, 2010
ISBN: 978-1430225492

[7] Allen Jones, Adam Freeman, *Visual C# 2010 Recipes: A Problem-Solution Approach*, Apress, 2010
ISBN: 978-1430225256

[8] Ben Watson, *C# 4.0 How-To*, Sams, 2010
ISBN: 978-0672330636

[9] Judith Bishop, *C# 3.0 Design Patterns*, O'Reilly, 2008
ISBN: 978-0596527730

[10] Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002
ISBN: 978-0321127426

[11] Martinovič Jan, *Architektura .NET přednáška, Diagnostika*, 2007
`http://arg.vsb.cz/msita/Courses/AT.NET/ppt/Instrumentation.zip`

[12] Microsoft Corp., *MSDN Library*, 3.4.2013
`http://msdn.microsoft.com/library/`

[13] Microsoft Corp., *Visual Studio Gallery*, 3.4.2013
`http://visualstudiogallery.msdn.microsoft.com/`